1. **Introducción**

En este taller se va a experimentar con modelos colaborativos para sistemas de recomendación utilizando un *dataset*de recomendación de música obtenido en <https://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/lastfm-1K.html>.

1. **Conocimiento del *dataset*de trabajo**

El *dataset* incluye la información de los hábitos de reproducción de los usuarios de *Last.fm[1]* de cerca de 1000 usuarios hasta el 5 de mayo del 2009, e información sobre los usuarios en dos tablas separadas: *userid-profile* y *userid-timestamp-artid-artname-traid-traname.tsv*.

Las entradas de datos de *userid-timestamp-artid-artname-traid-traname.tsv*relacionan un usuario con una fecha y hora (una marca temporal o un *timestamp*), un artista y una canción. El formato en el que están los datos es .tsv. La organización por columnas es <id del usuario, *timestamp*, id del artista, id de la canción>, donde el identificador tanto del artista como de la canción es un MBID.  Un MBID significa *MusicBrainz Identifiers*y cada MBID es único para una canción o un artista. La siguiente tabla muestra un breve perfilamiento de los datos:

|  |  |  |
| --- | --- | --- |
| Columna | ¿Contiene nulos? | Tipo de valores |
| userId | No | Cadena de texto |
| timestamp | No | Cadena de texto |
| artistId | Sí | Cadena de texto |
| artistName | Sí | Cadena de texto |
| trackId | Sí | Cadena de texto |
| trackName | Sí | Cadena de texto |

Tabla 1: Breve perfilamiento de datos

La primera decisión que se tomó al preprocesar los datos es eliminar las filas que contengan nulos en alguno de los identificadores, ya que es lo que se utiliza para generar el sistema de filtro colaborativo. En total se eliminaron XXXX filas por este motivo. También se eliminaron las filas que presentaran problemas de formato (más separadores de los necesarios, etc.). 9 filas se eliminaron de esta manera.

Esta tabla no incluye un *rating*del usuario a la canción o al artista, es únicamente información de hábitos de reproducción; un ejemplo se muestra a continuación.

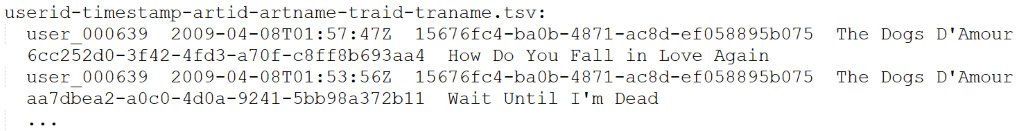


Figura 1. Ejemplo de fila de los datos en *userid-timestamp-artid-artname-traid-traname.tsv*.

Hay 83905 artistas diferentes que para efectos del modelo de sistemas de recomendación, son los ítems a recomendar.

La información de la tabla en *userid-profile*incluye información de los usuarios y un breve perfilamiento de ellos. Incluye el *id*del usuario, género, edad, país y fecha de registro. A continuación, se muestra el perfilamiento de estos datos y un ejemplo:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Columna | ¿Contiene nulos? | Número de valores no nulos | Número de valores únicos | Tipo de valores |
| #id | No | 992 | 992 | Valores consecutivos desde user\_000001 hasta user\_001000, con algunos faltantes |
| gender | Sí | 884 | 2 | M o F (o vacío) |
| age | Sí | 286 | 34 | Número entre 3 y 103, con algunos faltantes y vacíos |
| country | Sí | 907 | 66 | Nombre del país. Hay varios países que solo tienen un usuario, 25% de los usuarios son de Estados Unidos, 14% del Reino Unido, 5,5% de Polonia. El resto están todos debajo de 4%. |
| registered | Sí | 984 | 625 | Fecha entre el 29 de octubre del 2002 y el 5 de noviembre del 2009, con algunos faltantes. |

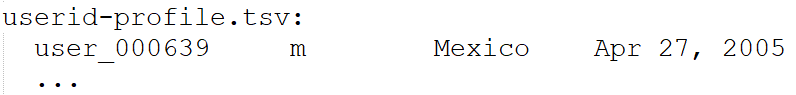


Figura 2. Ejemplo de fila en los datos en *userid-profile*

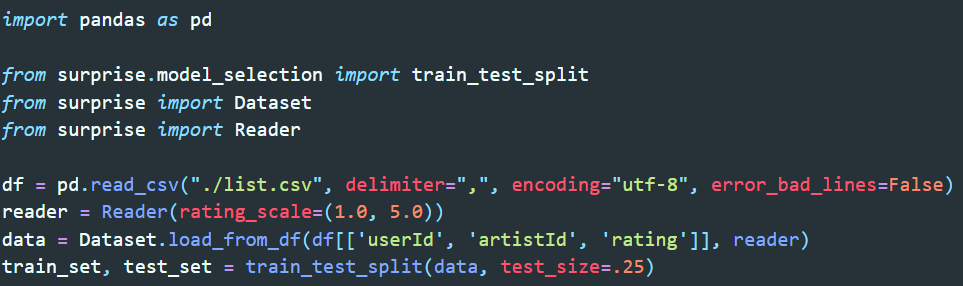
1. **Pre-procesamiento de datos**

Como no se tienen *ratings,*se van a definir según la interacción que tenga un usuario con los artistas. Para ello, se definió utilizar una función logarítmica que incluye el artista que más fue escuchado, de la forma: RatingArtista = LogMaxReproducido(ReproduccionesArtista) \* 4 + 1, si ReproduccionesArtista no es 0; en caso contrario, dejar este valor como 0 (esto ocurre cuando el usuario nunca ha escuchado al artista).

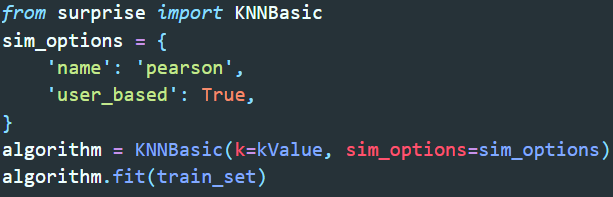
Esta fórmula devuelve un número entre 1.0 y 5.0. Se decidió así para no tener un comportamiento lineal que se vea sesgado a un artista con muchas reproducciones. Por ejemplo, usuario a escuchó el artista x un total de 1000 veces, y escuchó al grupo y un total de 50 veces. Si bien 50 reproducciones son relevantes, en una escala lineal se le daría un rating muy bajo.

Luego se formó una matriz donde cada columna es un usuario y cada fila un artista, y el contenido de la tabla es el rating que le da un usuario a un artista dado. Dado que esta matriz contiene muchos valores en cero, se formó una nueva matriz de tres columnas *userId*, *artistId*, *rating*, que reduce considerablemente el espacio necesario para almacenarla y sirve como entrada para un sistema de recomendación en la librería *Surprise*.

Los datos se separaron en dos conjuntos luego del pre-procesamiento, 25% de los datos se reservaron como *dataset*de pruebas. Esta separación se realiza con las siguientes instrucciones en *Surprise*, específicamente ***surprise.model\_selection.train\_test\_split***.



1. **Construcción de modelos colaborativos usuario usuario**
2. El modelo se construyó utilizando la librería surprise de Python. Para construir el modelo, se debe definir un algoritmo de predicción, en nuestro caso KNNBasic. KNNBasic recibe como parámetros el número máximo de vecinos, el nombre de una función de similitud y si es usuario-usuario o ítem-ítem.



1. Para predecir la relevancia de los ítems se utiliza la función *test*.

C:\Users\Rogelio Garcia\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\D1D59B0A.tmp

Esta función arroja predicciones con el identificador de usuarios *uid*, identificador del ítem *iid*, el *rating*real y el estimado, y otras informaciones. También incluye un campo para saber si fue posible predecir el rating para el ítem dado.

Prediction(uid='user\_000500', iid='2870dffb-f336-4a7b-96c0-eecf417c7fed', r\_ui=1.0, est=1.8971555182565383, details={'was\_impossible': True, 'reason': 'User and/or item is unkown.'})

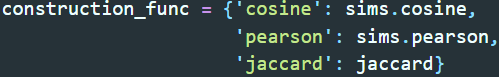
Prediction(uid='user\_000722', iid='d4a1404d-e00c-4bac-b3ba-e3557f6468d6', r\_ui=3.049423276345537, est=1.886442260279148, details={'actual\_k': 5, 'was\_impossible': False})

Adicionalmente, se definió una función que recibe las predicciones y un número entero n y retorna un diccionario que contiene una lista con las primeras n predicciones para cada usuario en orden del rating predicho, en forma de tupla (*itemId*, *rating*).

'user\_000463': [('32b90c92-9978-4a07-90eb-caa4b22f4907', 3.5106869703792136), ('d4d17620-fd97-4574-92a8-a2cb7e72ce42', 3.464256641763165), ('9de8f66e-3cd1-4f11-8328-38200f0612b0', 3.1446878961364217), ('cc0b7089-c08d-4c10-b6b0-873582c17fd6', 2.9901818013363446), ('f181961b-20f7-459e-89de-920ef03c7ed0', 2.728503442821299), ('8c538f11-c141-4588-8ecb-931083524186', 2.6670920880573963), ('970fb29f-e288-403e-a388-d2a7889bfa47', 2.643524734521892), ('8434409e-baa9-4e12-b4aa-566a91c7d7cf', 2.480210102296422), ('4efa55ba-93cf-497f-baf3-2ca9da7e193e', 2.4473736580596714), ('77f049ad-f469-4ad1-8283-7a2606a6722e', 2.436338420952059)]

Para implementar los tres índices de similitud, y para algunas partes de este taller, se volvió a implementar el modelo *KNNBasic*. La clase *KNNBasic* hereda de *SymmetricAlgoJaccard*, que a su vez hereda de *AlgoBase*. Para poder solucionar varios puntos del taller, se implementó *KNNBasic* con una ligera modificación para retornar vecinos según un umbral, y se implementó *SymmetricAlgoJaccard* con una modificación que anula o sobreescribe la función *compute\_similarities*, que retorna una matriz de similitud usuario-usuario (o ítem-ítem).

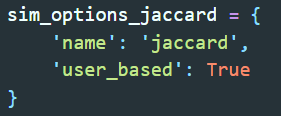
Todos los modelos utilizados son creados con la clase *KNNBasic* con un parámetro *sim\_*options diferente, que se traduce a una implementación diferente de *compute\_similarities*. Esta función utiliza otra función que calcula la matriz de similitud entre cada usuario (o cada ítem). Coseno y Pearson son importados de *similarities*, mientras que jaccard fue implementado en *jaccard\_sim.py*.



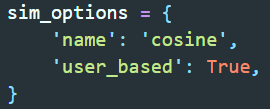
1. Para el índice de Jaccard fue necesario definir una nueva función de similitud. Esta similitud se calcula como:

De esta manera, durante el cálculo solo se mantiene una cuenta del número de artistas a los que ha calificado el usuario y la cuenta del número de artistas que los dos usuarios.

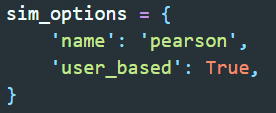
Al crear el modelo, se utiliza el siguiente parámetro:



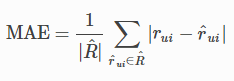
1. Un modelo basado en distancias coseno es fácil de lograr con *Surprise*. El parámetro *sim\_options* que entra como parámetro en *KNNBasic* lo define con la opción name.

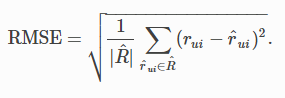


1. Un modelo basado en correlación de Pearson también es fácil de lograr con *Surprise*:

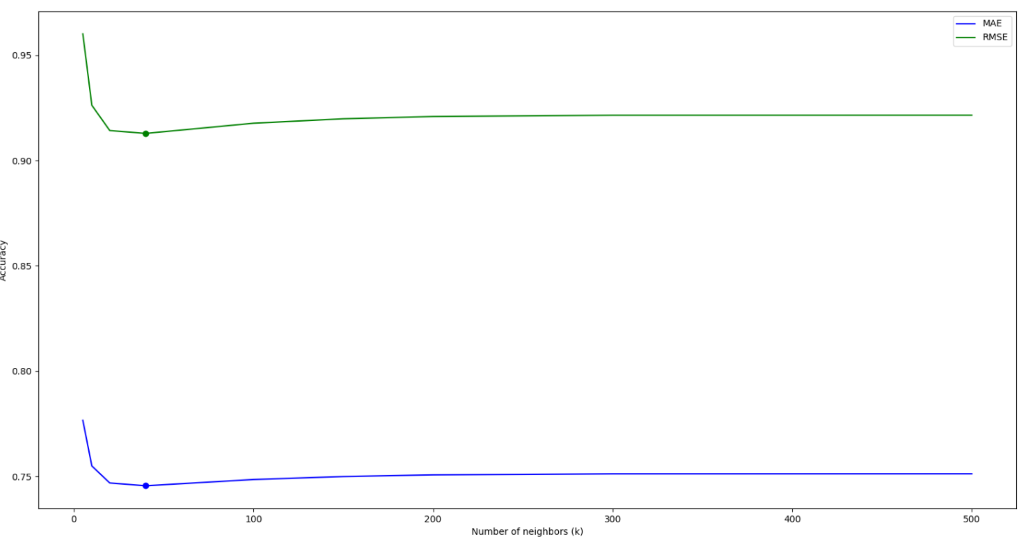


1. Para evaluar el comportamiento del modelo y cómo se está ajustando a los datos, se utilizaron las métricas RMSE (raíz del error cuadrático medio) y MAE (error absoluto medio).



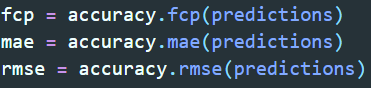


Experimentalmente se encontró que utilizando cualquiera de estas métricas los valores óptimos no varían demasiado. La siguiente figura muestra la determinación del número de vecinos óptima para el caso de la función de similitud de Pearson; varía en menos de 5 vecinos el valor óptimo.

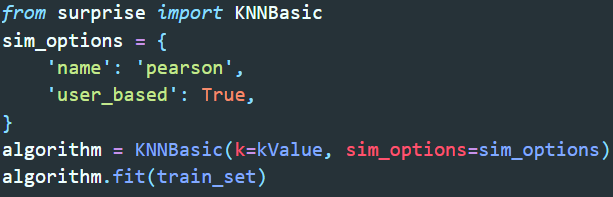


Conceptualmente, RMSE castiga más a un modelo por contener predicciones muy alejadas de los valores reales. Como no se quiere recomendar un ítem demasiado alejado de los gustos de la persona, se optó por utilizar RMSE para evaluar los diferentes modelos colaborativos.

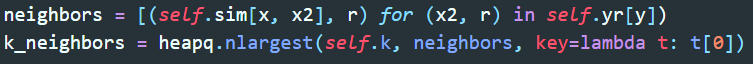
En *Surprise*se pueden calcular tres métricas diferentes: FCP (fracción de parejas concordantes), MAE y RMSE. Cada una se puede calcular así:



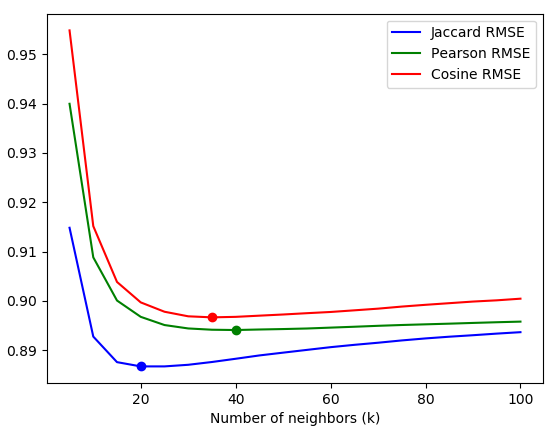
1. Se diseñó una estrategia para variar el parámetro del número de vecinos al definir el algoritmo a utilizar. En este caso es el valor *kValue*. Se itera sobre este valor y se analizan los resultados cada vez, guardando el valor para *kValue* que menor error obtuvo, por cada algoritmo analizado.



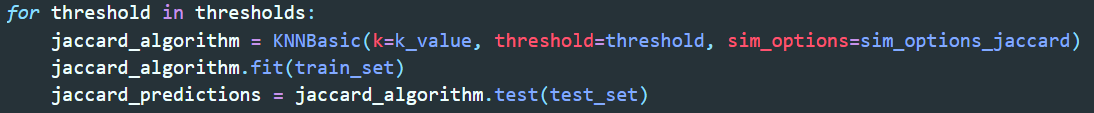
En la función *estimate*, de la clase *KNNBasic*, se forma un montículo con todos los vecinos y se retorna los *k* primeros, de la siguiente manera:



Se experimentó con un arreglo de *k\_*values de 5 a 100, en intervalos de 5. Para los tres índices de similitud utilizados hay un valor óptimo para el número de vecinos en el que el error es mínimo. A mayor o menor número de vecinos, el error aumenta. Graficamos el número óptimo de vecinos, y para los tres algoritmos está entre 20 y 40, con el menor error siendo de Jaccard.



También se definió una estrategia para analizar los resultados por umbral de similitud. Para ello, en *KNNBasic* se definió un nuevo parámetro opcional *threshold* con valor de 0 por defecto, y al seleccionar los vecinos de un usuario en la función *estimate*, utilizará solo la lista de los vecinos cuya similitud supere un cierto umbral:





Se experimentó con un arreglo de thresholds entre 0 y 1 en intervalos de 0.1 y se graficó el resultado. El resultado se graficó de la misma manera, mostrando a Jaccard con el menor error:

A close up of a map

Description automatically generated

1. La fórmula de McLaughlin fue implementada de la misma manera en que se implementó Jaccard